Comparación de Arquitecturas HPC para Computar Caminos Mínimos en Grafos. Intel Xeon Phi KNL vs NVIDIA Pascal

Manuel Costanzo[1], Enzo Rucci[1] *, Ulises Costi[2], Franco Chichizola[1], and Marcelo Naiouf[1]

[1] III-LIDI, Facultad de Informática, UNLP – CEA CICPBA.
La Plata (1900), Bs As, Argentina
{mcostanzo,erucci,francoch,maniouf}@lidi.info.unlp.edu.ar
[2] Facultad de Informática, UNLP.
La Plata (1900), Bs As, Argentina
ulises_92_lp@hotmail.com

Resumen En la actualidad, uno de los principales desafíos de los sistemas de cómputo de alto rendimiento consiste en mejorar su rendimiento manteniendo el consumo energético en niveles aceptables. En ese sentido, una estrategia consolidada consiste en emplear aceleradores como las GPUs o los procesadores many-core Intel Xeon Phi. En este trabajo, se describen y comparan dispositivos de las arquitecturas NVIDIA Pascal e Intel Xeon Phi Knights Landing. Seleccionando el algoritmo de Floyd-Warshall como caso representativo de aplicaciones de grafos y limitadas por memoria, se desarrollaron implementaciones optimizadas con el fin de analizar y comparar el rendimiento y la eficiencia energética en ambos dispositivos. Contrariamente a lo considerado en el análisis preliminar, se encontró que el rendimiento de ambos dispositivos resultó comparable mientras que, respecto a la eficiencia energética, el Xeon Phi se mostró superior.

Keywords: Caminos mínimos · Floyd-Warshall · Xeon Phi · Knights Landing · NVIDIA Pascal · Titan X

1. Introducción

En la última década, la búsqueda por mejorar la eficiencia energética de los sistemas de cómputo de altas prestaciones (HPC, por sus siglas en inglés) ha incentivado la tendencia hacia la computación heterogénea y las arquitecturas masivamente paralelas [6]. Los sistemas heterogéneos combinan CPUs con aceleradores (como pueden ser las GPUs de NVIDIA y AMD o los coprocesadores many-core Xeon Phi de Intel), delegando en estos las secciones de código con alta demanda computacional. De acuerdo con el ranking Green500 [3], la edición de

* Corresponding author.
[3] https://www.top500.org/green500/

Noviembre de 2010 contó con 17 sistemas que integraban aceleradores. Sin embargo, 10 años más tarde, este número se ha incrementado a 145, evidenciando la gran popularidad de esta estrategia.

En la actualidad, se puede considerar a las GPUs como la clase de acelerador dominante debido a su alta potencia de cómputo y eficiencia energética, además de su bajo costo de adquisición. En sentido opuesto, uno de sus puntos débiles es la necesidad de aprender un lenguaje específico para poder aprovecharlas al máximo, como pueden ser CUDA y OpenCL. Entre las empresas fabricantes, NVIDIA se destaca como la mayor proveedora para el segmento de alto rendimiento.

Por su parte, Intel ha presentado recientemente la segunda generación de sus procesadores Xeon Phi, con nombre clave Knigths Landing (KNL). A diferencia de su antecesor Knights Corner (KNC), KNL puede operar como un procesador autónomo. Entre sus principales características, se pueden mencionar la gran cantidad de núcleos con soporte para hyper-threading, la incorporación de las instrucciones vectoriales AVX-512 de 512 bits y la integración de una memoria de alto ancho de banda (HBM, por sus siglas en inglés), entre otras [10]. Más allá de la generación, lo que distingue a esta familia es la compatibilidad con arquitecturas x86, lo que les permite a lo programadores emplear modelos tradicionales en HPC, como OpenMP y MPI.

Debido a que cada acelerador tiene sus ventajas y desventajas para determinadas clases de problemas [16,2,13], la selección de la mejor opción para una aplicación dada se transforma en una cuestión fundamental en la búsqueda de lograr el máximo rendimiento. De manera de poder proveer algunas reglas generales para dicha selección, este artículo presenta un análisis de rendimiento entre dos arquitecturas HPC diferentes (Intel Xeon Phi KNL vs NVIDIA Pascal). Como caso de estudio, se seleccionó el algoritmo de Floyd-Warshall (FW) para el cómputo de caminos mínimos en un grafo, como caso representativo de aplicaciones de grafos y que se encuentren limitadas por memoria [12]. Se espera que el análisis presentado sea de ayuda para equipos de desarrollo al momento de elegir la arquitectura más conveniente para sus aplicaciones.

El resto del trabajo se organiza de la siguiente forma. En la Sección 2 se presentan los antecedentes y trabajos relacionados a la presente investigación. A continuación, en la Sección 3 se describen las implementaciones usadas mientras que en la Sección 4 se detalla el trabajo experimental realizado y se analizan los resultados obtenidos. Finalmente, en la Sección 5, se presentan las conclusiones y posibles líneas de trabajo futuro.

2. Antecedentes y Trabajos Relacionados

En primer lugar, se describen y comparan brevemente las arquitecturas Intel Xeon Phi KNL y NVIDIA Pascal. Luego, se describe el algoritmo FW para cómputo de caminos mínimos en un grafo. Finalmente, se detallan algunos trabajos relacionados al presente artículo.

2.1. Intel Xeon Phi KNL vs NVIDIA Pascal

Intel Xeon Phi KNL. A diferencia de una GPU o un co-procesador como el KNC, el KNL se trata de un procesador autónomo, capaz de bootear sistemas operativos y acceder directamente a la memoria DDR. La unidad de replicación escalable de la arquitectura KNL es el *tile*. Cada tile alberga 2 núcleos, una caché L2 compartida entre ambos núcleos, y una porción del directorio distribuido. Los núcleos de un tile implementan *simultaneous multithreading* (4 hilos hw por núcleo) y ejecución fuera de orden en su pipeline, además de contar con 2 unidades vectoriales que soportan las nuevas instrucciones AVX-512 de 512 bits [10].

Un chip de KNL esta conformado por entre 32 y 36 tiles activos (entre 64 y 72 núcleos), según el modelo en particular. Los tiles están interconectados por una malla 2D, con coherencia de cache basada en directorio distribuido y protocolo MESIF. Esta malla puede ser configurada en cinco modos de ejecución distintos (modos cluster). Según el modo de ejecución seleccionado, se definirá cómo el directorio distribuido se repartirá entre los tiles a lo largo del chip, lo que impacta en la latencia y el ancho de banda en el acceso a memoria.

El KNL trae una HBM de 16GB llamada MCDRAM, la cual está integrada en el mismo paquete del procesador. Esta memoria puede ser configurada en 3 modos distintos. En el modo *flat*, el espacio de direcciones es mapeado entre las dos memorias (MCDRAM y DDR), de modo que es el programador quien tiene la responsabilidad de definir cómo usarlas. Por su parte, el modo *caché* deja que el sistema se encargue de administrar la MCDRAM como una caché de la DDR. Por último, el modo *hybrid* asigna una parte de la MCDRAM como flat, y otra como cache [1].

NVIDIA Pascal. Pascal es la penúltima microarquitectura presentada por NVIDIA para el segmento de alto rendimiento, sucesora de Maxwell. En este segmento, una GPU de esta familia puede contar con hasta 3840 núcleos CUDA distribuidos entre (a lo sumo) 60 Streaming Multiprocessors (SM). A comparación de su predecesora, Pascal duplica la cantidad de registros por núcleo y aumenta el tamaño disponible de memoria compartida.

Esta microarquitectura presenta varias mejoras significativas respecto a Maxwell. Entre ellas, se puede destacar la incorporación en algunos de sus chips de una HBM de hasta 16GiB, que les permite alcanzar un ancho de banda de hasta 720 GB/s. También el reemplazo del tradicional bus PCIe para la comunicación CPU-GPU por un bus de alta velocidad (NVLink), que mejora significativamente la velocidad de comunicación. Por último, la provisión de una *memoria unificada*, consistente de un espacio de direccionamiento virtual entre las memorias de la CPU y la GPU, con el propósito de simplificar la programación [9].

Respecto al pico de rendimiento, algunos de los chips Pascal pueden alcanzar tasas de rendimiento en precisión doble (DP) de la mitad de precisión simple (SP). En sentido contrario, pueden duplicar el rendimiento de SP si computan en precisión media (PM) [4].

Tabla 1. Comparación entre Intel Xeon Phi KNL 7230 y NVIDIA Titan X

Dispositivo	**Intel Xeon Phi KNL**	**NVIDIA Titan X**
Chip	7230	GP102
Frecuencia	1.3-1.5 GHz	1.42-1.53 GHz
Cores	64 (256 hilos hw)	28 SMs (3584 cores CUDA)
Cache	1 MB L2	3 MB L2
SIMD	512-bit	-
HBM	16GB MCDRAM (450 GB/s)	-
Memoria RAM	192GB DDR4 (115.2 GB/s)	12GB GDDR5X (480.4 GB/s)
Bus	-	PCI-Express 3.0 x16
Rend. pico SP (DP)	6 (3) TFLOPS	10.97 (0.342) TFLOPS
TDP	215W	250W
TFLOPS/W (SP/DP)	0.028 / 0.014	0.051 / 0.001
Fecha de lanzamiento	Junio de 2016	Agosto de 2016

Breve comparación. La Tabla 1 presenta una comparación de las arquitecturas estudiadas y, en particular, considera los modelos utilizados en el trabajo experimental. Desde el punto de vista del pico de rendimiento teórico en SP, la Titan X resulta bastante superior al KNL 7230 (10.97 TFLOPS vs 6 TFLOPS). Sin embargo, debido al débil soporte de la primera para DP, es el KNL en este caso el que tiene una notable superioridad (3 TFLOPS vs 0.342 TFLOPS).

Respecto a la memoria principal, el KNL cuenta con una HBM que le permite estar casi a la altura de la Titan X en cuanto a ancho de banda, ya que el primero cuenta con tecnología DDR4 mientras que la segunda con GDDR5X. Sin embargo, por ser un coprocesador, la Titan X cuenta con un tamaño mucho menor de memoria respecto al KNL, que es un *host* en sí mismo.

Por último, al considerar la eficiencia energética (teórica), si bien la Titan X tiene una mayor potencia de diseño término (TDP), su ratio TFLOPS/W en SP casi duplica al del KNL por su mayor pico de rendimiento teórico. Por el contrario, el pobre desempeño de la Titan X para DP, lleva a que el KNL sea ampliamente superior en este caso.

A pesar de que ya han pasado algunos años de su lanzamiento, ambas arquitecturas se mantienen vigentes, como lo muestra la última edición del ranking Top500 [4] donde 17 sistemas están equipados con Xeon Phi KNL y otros 30 con GPUs de la familia Pascal.

2.2. Cómputo de Caminos Mínimos en un Grafo

Algoritmo FW. El pseudo-código de FW se muestra en la Fig. 1. Dado un grafo G de N vértices, FW recibe como entrada una matriz densa D de $N \times N$ que contiene las distancias entre todos los pares de vértices G, donde $D_{i,j}$ representa la distancia del nodo i al nodo j [5]. FW computa N iteraciones, evaluando en la

[4] Top500 `www.top500.org`

[5] Si no existe un camino entre los nodos i y j, se asigna *infinito* a su distancia (usualmente representado como el valor positivo más grande)

k-ésima iteración todos los posibles caminos entre los vértices i y j que tienen a k como vértice intermedio. Como resultado, produce una matriz actualizada D, donde $D_{i,j}$ contiene ahora la distancia mínima entre los nodos i y j hasta ese paso. Complementariamente, FW va construyendo una matriz adicional P que registra los caminos asociados a las distancias mínimas.

Algoritmo FW por bloques. A primera vista, la estructura de triple bucle anidado de este algoritmo resulta similar al de la multiplicación de matrices densas (MM). Sin embargo, como las lecturas y escrituras se realizan sobre la misma matriz, los tres bucles no pueden ser intercambiados libremente, como es el caso de MM. A pesar de esto, el algoritmo FW puede ser computado por bloques bajo ciertas condiciones [15].

```
for k ← 0 to N − 1 do
  for i ← 0 to N − 1 do
    for j ← 0 to N − 1 do
      if D_{i,j} ≥ D_{i,k} + D_{k,j} then
        D_{i,j} ← D_{i,k} + D_{k,j}
        P_{i,j} ← k
      end if
    end for
  end for
end for
```

Figura 1. Pseudocódigo del algoritmo de FW

El algoritmo FW por bloques (FWB) divide a la matriz D en bloques de tamaño $TB \times TB$, totalizando $(N/TB)^2$ bloques. El cómputo se organiza en $R = N/TB$ rondas, donde cada ronda consta de 4 fases ordenadas de acuerdo a las dependencias de datos entre los bloques:

1. Fase 1: actualizar el bloque $D^{k,k}$ debido a que sólo depende de sí mismo.
2. Fase 2: actualizar los bloques de la fila k de bloques ($D^{k,*}$) debido a que cada uno de estos depende de sí mismo y de $D^{k,k}$.
3. Fase 3: actualizar los bloques de la columna k de bloques ($D^{*,k}$) debido a que cada uno de estos depende de sí mismo y de $D^{k,k}$.
4. Fase 4: actualizar los bloques restantes $D^{i,j}$ de la matriz porque cada uno depende de los bloques $D^{i,k}$ y $D^{k,j}$ de su fila y columna de bloques, respectivamente.

En la Fig. 2 se muestra gráficamente cada una de las fases de cómputo y las dependencias entre bloques. Los recuadros en verde son aquellos bloques

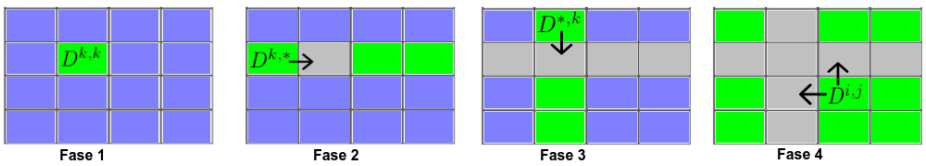

Figura 2. Fases del cómputo de FWB y sus dependencias

que están siendo computados, los grises los que ya se procesaron, y los celestes representan los que faltan computar.

2.3. Trabajos Relacionados

La comparación de arquitecturas HPC en un tema ampliamente estudiado por su comunidad. En particular, existen diversos artículos que involucran estudios comparativos entre Xeon Phi KNL y NVIDIA Pascal en el ámbito del modelado económico a gran escala [14], del álgebra lineal [3], de la dinámica de fluidos computacional [11], o del aprendizaje automático profundo [5], entre otros. Sin embargo, hasta donde llega el conocimiento de los autores, este es el primero en considerar algoritmos de grafos, en particular el de FW para caminos mínimos entre todos los vértices.

3. Optimización de FW

3.1. Implementación en Xeon Phi KNL 7230

La implementación usada considera las siguientes optimizaciones:

- *Localidad de datos.* Al computar según FWB, no sólo es posible explotar la localidad de datos sino que al mismo tiempo se logra aumentar el paralelismo disponible en la aplicación.
- *Paralelismo a nivel de hilos.* A partir del uso de OpenMP se obtiene una versión multi-hilada. Tanto los bloques de la fase 2 como los de la fase 3 son repartidos entre los diferentes hilos mediante la directiva `for` con planificación `dynamic`. En el caso de la fase 1, al consistir de un único bloque, los hilos se reparten las iteraciones dentro del mismo.
- *Paralelismo a nivel de datos.* A través de la directiva OpenMP `simd` se vectorizan las operaciones del bucle más interno en el cómputo de cada bloque, lo que permite aprovechar las instrucciones AVX-512.
- *Loop unrolling.* Desenrollando en forma completa el bucle más interno y una única vez el bucle i.
- *Predicción de saltos.* Mediante la inclusión de la función integrada del compilador `__builtin_expect`, se lo ayuda a predecir mejor los saltos de la sentencia `if`.
- *MCDRAM.* Al ser una aplicación limitada por ancho de banda, se obtienen grandes beneficios por el uso de esta memoria especial. Las ejecuciones se realizan usando el comando `numactl`.

Resulta importante aclarar que se puede considerar a esta implementación como una versión optimizada de [12], ya que incluye además la paralelización intra-bloque de la fase 1 y la mejora en la predicción de saltos.

3.2. Implementación en NVIDIA Titan X

La implementación utilizada considera las optimizaciones conocidas para FW al momento [7,8]. Entre ellas se pueden mencionar:

- *Concurrencia*. Se han desarrollado 3 kernels que son invocados una vez por cada ronda de cómputo de FWB. En la ronda k, el primer kernel computa el bloque $D^{k,k}$ (fase 1) y se instancia con un único grid formado por un único bloque. A continuación, se invoca el segundo kernel que computa en forma combinada los bloques de las fases 2 y 3 ($D^{k,*}$, $D^{*,k}$), el cual se instancia con un único grid de $2 \times (R-1)$ bloques. Por último, se invoca al tercer kernel que computa los bloques restantes correspondientes a la fase 4 ($D^{i,j}$). Este kernel se instancia con un único grid de $(R-1)^2$ bloques. En todos los casos, los bloques se conforman de $TB \times TB$ hilos.
- *Explotación de jerarquía de memoria*. Para el cómputo de FWB, el uso de memoria compartida no sólo resulta conveniente sino también necesario, especialmente en las fases 1-3 ya que los hilos leen y escriben en los mismos bloques de la matriz debido a sus dependencias. En el caso de la fase 4, es posible aprovechar la memoria local para el bloque de escritura ($D^{i,j}$), lo que mejora aun más los tiempos de acceso. Por último, los accesos a memoria principal fueron organizados de forma de que sean coalescentes.
- *Ocupación de recursos*. De forma de maximizar este aspecto, se varió el tamaño de los bloques de hilos usando $TB = \{8, 16, 32\}$ para encontrar aquel que conduce al máximo número posible de *warps* activos.
- *Loop unrolling*. Desenrollando en forma completa el bucle que computa cada hilo.

4. Resultados Experimentales

4.1. Diseño experimental

Las pruebas se realizaron en dos plataformas diferentes [6]. Por un lado, un servidor Intel Xeon Phi KNL 7230 configurado en modo cluster *All-to-all* y de memoria *flat* (ICC v19.0.0.117). Por otro, un equipo Intel Core i7-7700 3.6 GHz y 16GB RAM, el cual integra una GPU NVIDIA Titan X (CUDA v9.0).

Para ambas plataformas se consideró la variación del tamaño de la matriz de distancias ($N = \{4096, 8192, 16384, 32768, 65536\}$) y el tipo de dato (*float*, *double*). En el caso del KNL, se varió tanto la cantidad de hilos OpenMP $T = \{64, 128, 192, 256\}$ como el $TB = \{16, 32, 64, 128\}$, encontrando como valores óptimos $T_{float} = 128$, $T_{double} = 64$ y $TB = 64$. En cuanto a la GPU, los mejores rendimientos se presentaron al usar $TB_{float} = 32$ y $TB_{double} = 16$. Por último, cada prueba particular fue repetida 15 veces, calculando los promedios para reducir la variabilidad.

[6] Las características de cada una fueron descritas al final de la Sección 2.1

4.2. Resultados de Rendimiento y Eficiencia Energética

Para evaluar el rendimiento se emplea la métrica GFLOPS, utilizando la fórmula $GFLOPS = \frac{2 \times N^3}{t \times 10^9}$, donde N es el tamaño de la matriz de distancias, t es el tiempo de ejecución (en segundos) y el factor 2 representa la cantidad de operaciones en punto flotante requerida por cada iteración del bucle más interno.

En la Fig. 3 se presenta el rendimiento obtenido por cada implementación al variar la magnitud de la matriz y el tipo de dato usado. En SP (*float*), se puede observar que la GPU obtiene mejores rendimientos para los tamaños de matriz más chicos, siendo aproximadamente 19 % superior cuando $N = 4096$. En estos casos, la GPU se adapta mejor en este caso que el KNL, el cual requiere cargas mayores de trabajo para alcanzar su máximo aprovechamiento. Este hecho se ve reflejado en el gráfico ya que a medida que se incrementa el tamaño de la matriz de distancias, es el KNL quien obtiene las mejores prestaciones alcanzando hasta un 7 % adicional de diferencia. Resulta importante notar que con $N = 65536$, el KNL experimenta una pérdida de rendimiento de aproximadamente 30 %, debiéndose al hecho de que con ese valor, se supera el tamaño disponible de la MCDRAM y se requiere del uso parcial de la DDR4, la cual posee un ancho de banda bastante menor. Aun así, resulta más conveniente que el caso de la GPU, quien no puede computar los casos con $N > 32768$ debido a que se supera el espacio disponible en la memoria principal [7].

En cuanto a DP (*double*), los resultados obtenidos son los esperables debido al débil soporte para esta clase de operaciones que tiene la Titan X. Mientras que la GPU obtiene un rendimiento casi constante de ∼90 GFLOPS, el KNL mejora su rendimiento a medida que N crece hasta superar el tamaño de la MCDRAM. En particular, el KNL obtiene cerca de la mitad de los FLOPS que en SP, pero que mejoran hasta 4.3× los de la Titan X. Por último, el problema del límite de memoria en la GPU, se ve agravado por el hecho de que el tipo *double* requiere más espacio (no fue posible computar los casos con $N > 16384$).

Como se mencionó en la introducción, el rendimiento de la aplicación no es el único punto de interés a considerar, sino que también importa la eficiencia energética. La Tabla 2 presenta los cocientes de eficiencia energética teniendo en cuenta los picos de GFLOPS alcanzados y el TDP de las plataformas utilizadas. Se puede notar que el KNL resulta superior en ambos casos. En particular, KNL obtiene un cociente GFLOPS/Watt que resulta 1.2× mejor que la Titan X en SP. Sin embargo, este factor crece hasta 5.5× en DP, debido al débil soporte para estas operaciones de esta GPU. Se debe notar que en este análisis no se consideró el consumo del *host* por parte de la GPU, con lo cual las diferencias podrían ser aún mayores.

[7] Por supuesto que es posible desarrollar una implementación que procese la matriz de a partes y no tenga esta limitación de memoria. Sin embargo, la necesidad de hacer operaciones de E/S para cada ronda, perjudicarían el rendimiento significativamente.

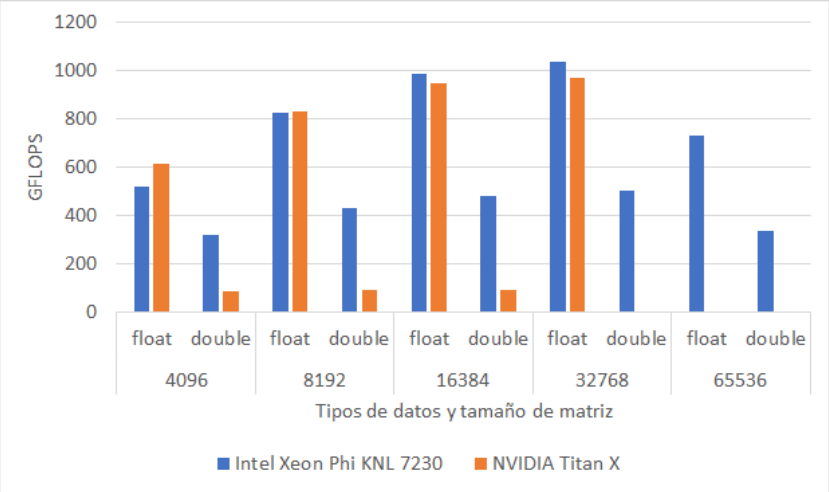

Figura 3. Rendimiento obtenido al variar el tipo de dato y el tamaño de la matriz de distancias en Intel Xeon Phi KNL 7230 y NVIDIA Titan X

Tabla 2. Comparación de rendimiento y eficiencia energética entre plataformas

Plataforma	GFLOPS (pico) SP	GFLOPS (pico) DP	TDP (Watt)	GFLOPS/Watt SP	GFLOPS/Watt DP
Xeon Phi KNL 7230	1037	501	215	4,82	2,33
NVIDIA Titan X	972	90	250	3,89	0,36

4.3. Limitaciones

Resulta importante mencionar que este análisis se centra en dos modelos particulares de estas arquitecturas y que, como tales, algunos de los resultados encontrados podrían cambiar si se emplearan otros. Por ejemplo, la GPU NVIDIA GP100 (Pascal) posee un pico de rendimiento en SP levemente menor al de la Titan X (10.1 TFLOPS). Sin embargo, su soporte para DP es ampliamente superior, siendo de la mitad de SP (5 TFLOPS). En ese sentido, se considera que la comparación resulta igualmente valiosa para mostrar tendencias y características distintivas de cada arquitectura, aun cuando existan diferentes modelos de cada una que puedan presentar variaciones en sus especificaciones.

5. Conclusiones y Trabajo Futuro

Este trabajo se enfoca en la comparación de arquitecturas Intel Xeon Phi KNL y NVIDIA Pascal. Tomando como caso de estudio el algoritmo FW para cómputo de caminos mínimos en un grafo, se utilizaron implementaciones opti-

mizadas para comparar el rendimiento alcanzable en cada plataforma y así poder extraer algunas reglas generales. Entre ellas se pueden mencionar:

- A pesar de que en el análisis preliminar la Titan X se mostraba ampliamente superior al KNL, los rendimientos (SP) obtenidos para FW fueron comparables. Mientras que la GPU obtuvo mejores prestaciones para grafos pequeños, a medida que se incrementó el tamaño de la matriz de distancias, fue el KNL el que obtuvo un rendimiento superior. De este hecho se desprende la necesidad del KNL de grandes cargas de trabajo para poder aprovecharlo al máximo.
- En cuanto a la eficiencia energética, contrariamente a lo encontrado en el análisis preliminar, fue el KNL el que obtuvo el mejor ratio en SP. Este resultado contribuye al hecho de que el rendimiento sostenible y la eficiencia energética de los dispositivos varían de acuerdo a cada problema particular y a su correspondiente implementación en software.
- Más allá del aprendizaje específico de sus lenguajes, las GPUs pueden requerir esfuerzos de programación adicionales. Aunque experimentó una pérdida de rendimiento significativa cuando el tamaño de la matriz superó el de la MCDRAM, el KNL fue capaz de procesar todos los grafos. Por su parte, la Titan X no fue capaz de procesar aquellos que superaran el tamaño de su memoria RAM. Si bien es posible desarrollar una implementación que sí lo haga, también tendría una pérdida de rendimiento asociada y requeriría de un esfuerzo adicional de programación (ausente en el caso del KNL).

Entre los trabajos futuros, se pueden mencionar:

- Extender la implementación GPU para que soporte grafos más grandes que el tamaño de la memoria principal.
- Medir el consumo real en ambos dispositivos para hacer un estudio más preciso de la eficiencia energética, considerando que el primero puede diferir del estimado.
- Incluir otros modelos de las arquitecturas en cuestión (especialmente de las GPUs Pascal).

El desarrollo de estas actividades le otorgaría mayor robustez y representatividad al estudio realizado.

Agradecimientos. Los autores agradecen el soporte de la empresa NVIDIA por la donación de la GPU Titan X usada en esta investigación.

Referencias

1. Codreanu, V., Rodríguez, J., Saastad, O.W.: Best Practice Guide - Knights Landing (2017), `https://bit.ly/2CEolbR`
2. Deng, L., Bai, H., Zhao, D., Wang, F.: Kepler gpu vs. xeon phi: Performance case study with a high-order cfd application. In: 2015 IEEE International Conference on Computer and Communications (ICCC). pp. 87–94 (2015)

3. Deveci, M., Trott, C., Rajamanickam, S.: Multithreaded sparse matrix-matrix multiplication for many-core and gpu architectures. Parallel Computing **78**, 33 – 46 (2018). https://doi.org/https://doi.org/10.1016/j.parco.2018.06.009, http://www.sciencedirect.com/science/article/pii/S0167819118301923
4. Foley, D., Danskin, J.: Ultra-performance pascal gpu and nvlink interconnect. IEEE Micro **37**(2), 7–17 (2017)
5. Gawande, N.A., Daily, J.A., Siegel, C., Tallent, N.R., Vishnu, A.: Scaling deep learning workloads: Nvidia dgx-1/pascal and intel knights landing. Future Generation Computer Systems **108**, 1162 – 1172 (2020)
6. Giefers, H., Staar, P., Bekas, C., Hagleitner, C.: Analyzing the energy-efficiency of sparse matrix multiplication on heterogeneous systems: A comparative study of gpu, xeon phi and fpga. In: 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). pp. 46–56 (2016)
7. Katz, G.J., Kider, Jr, J.T.: All-pairs shortest-paths for large graphs on the gpu. In: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware. pp. 47–55. GH '08, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2008)
8. Lund, B.D., Smith, J.W.: A multi-stage CUDA kernel for floyd-warshall. CoRR **abs/1001.4108** (2010), http://arxiv.org/abs/1001.4108
9. NVIDIA: NVIDIA Tesla P100., https://bit.ly/2Ozrrk1
10. Reinders, J., Jeffers, J., Sodani, A.: Intel Xeon Phi Processor High Performance Programming Knights Landing Edition. Morgan Kaufmann Publishers Inc., Boston, MA, USA (2016)
11. Robertsén, F., Mattila, K., Westerholm, J.: High-performance simd implementation of the lattice-boltzmann method on the xeon phi processor. Concurrency and Computation: Practice and Experience **31**(13) (7 2019). https://doi.org/10.1002/cpe.5072
12. Rucci, E., De Giusti, A., Naiouf, M.: Blocked all-pairs shortest paths algorithm on intel xeon phi knl processor: A case study. In: De Giusti, A.E. (ed.) Computer Science – CACIC 2017. pp. 47–57. Springer Int. Pub., Cham (2018)
13. Rucci, E., Garcia, C., Botella, G., De Giusti, A., Naiouf, M., Prieto-Matias, M.: Swifold: Smith-waterman implementation on fpga with opencl for long dna sequences. BMC Systems Biology **12**(5), 96 (Nov 2018). https://doi.org/10.1186/s12918-018-0614-6
14. Scheidegger, S., Mikushin, D., Kubler, F., Schenk, O.: Rethinking large-scale economic modeling for efficiency: Optimizations for gpu and xeon phi clusters. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 610–619 (2018)
15. Venkataraman, G., Sahni, S., Mukhopadhyaya, S.: A Blocked All-Pairs Shortest-Paths Algorithm, pp. 419–432. Springer Berlin Heidelberg (2000). https://doi.org/10.1007/3-540-44985-X_36
16. Véstias, M., Neto, H.: Trends of cpu, gpu and fpga for high-performance computing. In: 2014 24th International Conference on Field Programmable Logic and Applications (FPL). pp. 1–6 (Sept 2014). https://doi.org/10.1109/FPL.2014.6927483